Querying Services Based on Composition Context

Ngoc Chan Nguyen, Walid Gaaloul

Querying Services Based on Composition Context

Nguyen Ngoc Chan
Université de Lorraine,
LORIA UMR 7503, France

Walid Gaaloul
TELECOM SudParis,
Samovar UMR 5157 CNRS, France

***Abstract*—Finding suitable services to be composed in a service-based business process is always a big challenge. In this paper, we present a query language that addresses this challenge by exploiting relations between services in process models. Concretely, we define the composition context of a service as a process fragment around it. We match composition contexts to compute the similarity between services. Finally, we conduct the query based on this computed similarity and flexibly added constraints. Our proposed query helps to retrieve services that have similar composition context with the queried service. It can be applied in process design or execution phase. We perform experiments on a large dataset of real process models. Experimental results show that our approach is feasible and efficient.**

I. Introduction

Services have been developed as loosely-coupled applications that can be composed in service-based processes to achieve different business goals. Designing service-based business process is a crucial task that enables companies to sketch out their business, preview the process in real time execution, determines opportunities and risks involved in their business [22], [20]. The importance of process design involves many frameworks and applications developed to facilitate this task. Most of recent efforts attempt to analyze business process topology, such as [3], [8], [1], [19], to assist process designer by finding business processes that are similar to an ongoing designed one.

However, finding similar processes is not always helpful as the process designer may have difficulty to be inspired by large-scale, complex processes. Moreover, in some circumstances, process designer needs to find services instead of whole business processes. For example, a process designer may want to find suitable services to plug into some specific positions to complete his design (see Fig. 1a). In another case, he may want to find alternatives of a service to replace it in case of failure (see Fig. 1b).

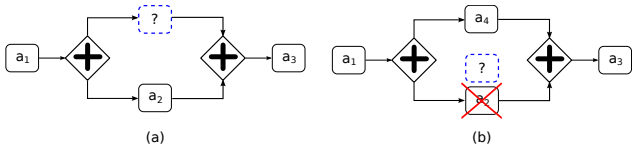

Fig. 1: Finding services to be composed in a business process

In this paper, we present an approach that assist process designers by finding services that are similar to a selected service's position in an ongoing designed process. We propose and develop a query language as a tool to search for similar services. We take existing process models as input in order to learn from the past design, not to reinvent the wheel. We exploit the knowledge acquired for previous designed process to infer service similarity. Concretely, similarity between services is computed based on their *composition context*, which is constructed based on relations between services in existing process models.

By providing a query language and focusing on service composition context, the objective of our approach is twofold: (i) to propose a new approach to compute the similarity of component services in services-based processes based on their composition context and (ii) to provide a useful query tool for service selection. Our approach does not only help to facilitate process design but also can improve the availability of services by identifying alternatives of services in order to replace them in case of failure.

Our paper is organized as follows. The next section elaborates our approach to compute the similarity between services based on their composition context. Section III presents our proposed query. Experiments are shown in section IV. We present the related work in section V and conclude our work in section VI.

II. Service Similarity Based on Composition Context

This section elaborates our approach to compute the similarity between two services based on their composition context. To illustrate our approach, we assume that a process designer is designing a train reservation process. Fig. 2 shows the incomplete process and the designer is looking for suitable services to plug into the missing position (a_x). We also assume that there exists a flight reservation process (Fig. 3) in the repository of designed processes. We demonstrate in the following the computation of similarity between service 'Request credit card Info.' (a_6) and the queried service (a_x).

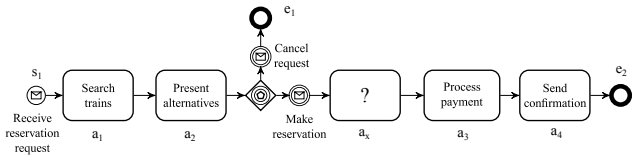

Fig. 2: Incomplete train reservation process

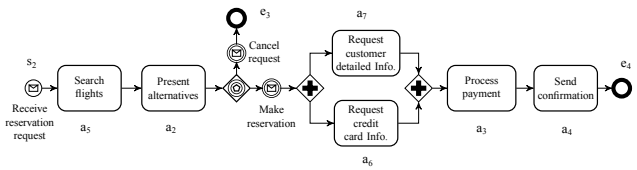

Fig. 3: Flight reservation process

A. Preliminaries

In our work, terminal events (start events, end events) are considered as terminal services. We define a *connection element* as either a *connecting object* (e.g. sequence, message flow), or a *gateway*, (e.g. AND-split, OR-split, etc.), or an *intermediate event* (e.g. error message, message-catching, etc.).

Let A_P be the set of services and C_P be the set of connection elements in a process P.

Definition II.1 (next relation)
Let $e_i, e_j \in A_P \cup C_P$. A next relation from e_i to e_j, denoted by $e_i \rightarrow_P e_j$, indicates that e_j situates right after e_i in P.

Definition II.2 (connection flow)
A connection flow from a_i to a_j, $a_i, a_j \in A_P$, denoted by ${}^{a_j}_{a_i}f_P$, is a sequence of connection elements $c_1, c_2, \ldots, c_n \in C_P$ that satisfies: $a_i \rightarrow_P c_1$, $c_1 \rightarrow_P c_2, \ldots, c_{n-1} \rightarrow_P c_n, c_n \rightarrow_P a_j$. ${}^{a_j}_{a_i}f_P \in C_P^*$, C_P^* is the set of sequences of connection elements in P.

For example, in Fig. 3, the connection flow from s_2 to a_5 or from a_5 to a_2 is 'flow-transition'; the connection flow from a_2 to e_3 is 'event-based-gateway''message-catching'; the connection flow from a_2 to a_7 is 'event-based-gateway''message-catching''parallel-split' and so on.

Definition II.3 (connection path)
A connection path from a_i to a_j, denoted by ${}^{a_j}_{a_i}\mathcal{P}_P$, is a sequence of services $a_1, a_2, \ldots, a_k$ where $a_1 = a_i$, $a_k = a_j$ and $\exists({}^{a_{t+1}}_{a_t}f_P \in C_P^* \ \vee \ {}^{a_t}_{a_{t+1}}f_P \in C_P^*)\ \forall 1 \leq t \leq k-1$.

Definition II.4 (connection path length)
The length of a connection path ${}^{a_j}_{a_i}\mathcal{P}_P$, denoted by $\mathcal{L}({}^{a_j}_{a_i}\mathcal{P}_P)$ is the number of connection flows in the path.

Definition II.5 (shortest connection path)
The shortest connection path between a_i and a_j, denoted by ${}^{a_j}_{a_i}\mathcal{S}_P$, is the connection path between them that has the minimum connection path length.

For example, in Figure 3, the shortest connection path from a_5 to a_7 is $a_5a_2a_7$ and its length is 2.

Definition II.6 (k^{th}-layer neighbor)
a_j is a k^{th}-layer neighbor of a_i in a process P iff $\exists {}^{a_j}_{a_i}\mathcal{P}_P : \mathcal{L}({}^{a_j}_{a_i}\mathcal{P}_P) = k$. The set of k^{th}-layer neighbors of a service a_i is denoted by $N_P^k(a_i)$. $N_P^0(a_i) = \{a_i\}$.

For example in Fig. 3, s_2 and a_2 are the 1^{st}-layer neighbors of a_5; a_5, e_3, a_7 and a_6 are the 1^{st}-layer neighbors of a_2; a_7 is one of the 2^{nd}-layer neighbors of a_5 and so on.

As the distance from a service a_i to its k^{th}-layer neighbors is k, we can imagine that the k^{th}-layer neighbors of a service a_i are located on a latent circle whose center is a_i and k is the radius. We call this latent circle a *connection layer* and the area limited by two adjacent latent circles a *connection zone*. Connection layers and connection zones of a service are numbered. A connection flow connecting two $(k-1)^{th}$-layer neighbors, or a $(k-1)^{th}$-layer neighbor to a k^{th}-layer neighbor is called a *k^{th}-zone flow* (Definition II.7).

Definition II.7 (k^{th}-zone flow)
${}^{a_v}_{a_u}f_P$ is a k^{th}-zone flow of a_i iff $\exists {}^{a_v}_{a_u}f_P : (a_u, a_v \in N_P^{k-1}(a_i)) \vee (a_u \in N_P^{k-1}(a_i) \wedge a_v \in N_P^k(a_i)) \vee (a_v \in N_P^{k-1}(a_i) \wedge a_u \in N_P^k(a_i))$. The set of all k^{th}-zone flows of a service $a_i \in P$ is denoted by $Z_P^k(a_i)$. $Z_P^0(a_i) = \emptyset$ and $|Z_P^k(a_i)|$ is the number of connection flows in the k^{th} connection zone of a_i.

For example in Fig. 3, the connection from a_2 to a_7 is a 2^{nd}-zone flow of a_5 while the connection from a_7 to a_3 is one of its 3^{rd}-zone flows. $|Z_{P_2}^2(a_5)| = 3$ as in the 2^{nd}-zone of a_5, there are three connection flows, which are from a_2 to e_3, a_7 and a_6.

B. Service composition context

Intuitively, the connection paths between two services in a service-based process present their relation in term of closeness. The longer the connection path is, the weaker their relation is and the shortest connection path between two services presents their best relation. To illustrate the best relations of a service to other services in a service-based process, we define the *composition context graph* (formally defined in Definition II.8) which presents all the shortest paths from a service to others.

Each service in a service-based process has a composition context graph. Each vertex in this graph is associated to a number which indicates *the shortest path length of the connection path to the associated service*. The vertices that have the same distance to the associated service are located on the same layer around the associated service. We name the number associated to each service the *layer number*. The area limited between two adjacent layers is called zone. The edge connecting two vertexes in a composition context graph belongs to a zone. We assign to each edge in the composition context graph a number, so-called *zone number*, which determines the zone that the edge belongs to.

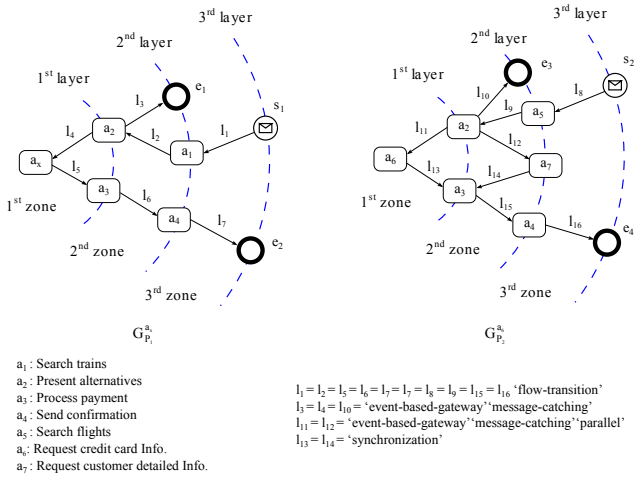

Fig. 4: Composition context graphs of a_x (in Fig. 2) and a_6 (in Fig. 3)

Definition II.8 (Composition context graph)
The composition context graph of a service $a_x \in P$, denoted by $G_P^{a_x} = (V_P^{a_x}, E_P^{a_x}, L_P^{a_x})$, is a labeled directed graph. $V_P^{a_x}$ is a set of vertexes associated to their layer numbers; $E_P^{a_x}$ is a set of edges associated to their zone numbers; and $L_P^{a_x}$ is a set of edge labels in $G_P^{a_x}$. $V_P^{a_x}$, $E_P^{a_x}$, and $L_P^{a_x}$ are defined as:

- $V_P^{a_x} = \{(a_i, \mathcal{L}({}^{a_x}_{a_i}\mathcal{S}_P)) : \ a_i \in V_P\}$

- $E_P^{a_x} = \{((a_i, a_j), {}^{a_j}_{a_i} z_P^{a_x}) : (a_i, a_j) \in E_P, {}^{a_j}_{a_i} z_P^{a_x} = Min(\mathcal{L}({}^{a_x}_{a_i}\mathcal{S}_P), \mathcal{L}({}^{a_x}_{a_j}\mathcal{S}_P)) + 1\}$
- $L_P^{a_x} = \{((a_i, a_j), {}^{a_j}_{a_i} f_P) : (a_i, a_j) \in E_P, {}^{a_j}_{a_i} f_P \in C_P^*\}$.

For example, the composition context graph of a_x in Fig. 2 is $G_{P_1}^{a_x} = (V_{P_1}^{a_x}, E_{P_1}^{a_x}, L_{P_1}^{a_x})$ where $V_{P_1}^{a_x}$ = $\{(a_x, 0)$, $(a_2, 1)$, $(a_3, 1)$, $(a_1, 2)$, $(e_1, 2)$, $(a_4, 2)$, $(s_1, 3)$, $(e_2, 3)\}$; $E_{P_1}^{a_x}$=$\{((a_2, a_x), 1)$, $((a_x, a_3), 1)$, $((a_2, e_1), 2)$, $((a_1, a_2), 2)$, $((a_3, a_4), 2)$, $((s_1, a_1), 3)$, $((a_4, e_2), 3)\}$; $L_{P_1}^{a_x}$ = L_{P_1}= $\{((s_1, a_1)$, 'flow-transition'), $((a_1, a_2)$, 'flow-transition'), $((a_2, e_1)$,'event-based-gateway''message-catching'), $((a_2, a_x)$, 'event-based-gateway''message-catching'), $((a_x, a_3)$, 'flow-transition'), $((a_3, a_4)$, 'flow-transition'), $((a_4, e_2)$, 'flow-transition')$\}$. This composition context graph and the composition context graph a_6 in Fig. 3 are depicted in Fig. 4.

C. Composition context graph matching

In this section, we present our approach to compute the matching between two composition contexts. Concretely, we *match all connection flows that belong to the same connection zone and have the same ending services*. Particularly, for the first zone, we *match the connection flows connecting the associated services to the same services in the first layer*.

1) Connection flow matching: We propose to apply Levenshtein distance (LD for short) [15] to compute the matching between connection flows. In information theory, the LD is a metric for measuring the difference between two sequences of characters. The LD is defined as the minimum number of edits needed to transform one sequence of characters into the other, with the allowable edit operations being *insertion*, *deletion*, or *substitution* of a single element.In our approach, we consider each connection element as a character and present *a connection flow as a sequence of characters*. Then, the similarity between two connection flows are inferred by the similarity between corresponding sequences of characters.

Concretely, given two connection flows ${}^{a_j}_{a_i} f_{P_1} = c_1 c_2 \dots c_n$ and ${}^{a_y}_{a_x} f_{P_2} = c'_1 c'_2 \dots c'_m$, the pattern matching between them is computed by Equation 1.

$$M({}^{a_j}_{a_i} f_{P_1}, {}^{a_y}_{a_x} f_{P_2}) = 1 - \frac{LD({}^{a_j}_{a_i} f_{P_1}, {}^{a_y}_{a_x} f_{P_2})}{Max(n, m)} \quad (1)$$

In our example, we have $M({}^{a_2}_{a_1} f_{P_1}, {}^{a_2}_{a_5} f_{P_2}) = M$('flow-transition', 'flow-transition') =1; $M_p({}^{a_x}_{a_2} f_{P_1}, {}^{a_6}_{a_2} f_{P_2}) = M$('event-based-gateway''message-caching', 'event-based-gateway' 'message-caching''parallel') = 0.67 and so on.

2) Context matching: To compute the composition context matching between two services, we propose to sum up the matching of the connection flows in the two contexts. There are two cases to consider: matching in the *first zone* and matching in *other zones*. In the first zone, we match the connection flows that connect the two associated services and same services in the first layer. In other zones, we match the connection flows that connect the same services. For example, to compute the context matching between a_x and a_6 (Figure 4), we will match $({}^{a_x}_{a_2} f_{P_1}, {}^{a_6}_{a_2} f_{P_2})$, $({}^{a_3}_{a_x} f_{P_1}, {}^{a_3}_{a_6} f_{P_2})$, $({}^{e_1}_{a_2} f_{P_1}, {}^{e_3}_{a_2} f_{P_2})$, $({}^{a_4}_{a_3} f_{P_1}, {}^{a_4}_{a_3} f_{P_2})$ and $({}^{e_2}_{a_4} f_{P_1}, {}^{e_4}_{a_4} f_{P_2})$.

Besides, the consumption context of a service is strongly reflected by the connection flows to its closest neighbors while the interactions with other neighbors in the further layers do not heavily reflect its consumption context. Therefore, we propose to assign a weight (w_t) for each t^{th} connection zone, so called *zone-weight* and integrate this weight into the similarity computation. Since the zone-weight has to have greater values in smaller t^{th} connection zone, we propose the zone-weight a value computed by a polynomial function which is $w_t = \frac{k+1-t}{k}$, where t is the zone number ($1 \leq t \leq k$) and k is the number of considered zones around the service. Then, the composition contexts matching between $G_{P_1}^{a_i}$ and $G_{P_2}^{a_j}$ within k zones, denoted by $\mathbb{MW}^k(G_{P_1}^{a_i}, G_{P_2}^{a_j})$, is given by Equation 2.

$$\mathbb{MW}^k(G_{P_1}^{a_i}, G_{P_2}^{a_j}) = \frac{2}{k+1} \times \sum_{t=1}^{k} \frac{k+1-t}{k} \times \frac{\sum_{\substack{{}^{a_v}_{a_u} f_{P_1} \in Z_{P_1}^t \\ {}^{a_n}_{a_m} f_{P_2} \in Z_{P_2}^t}} \mathsf{MF}^t({}^{a_v}_{a_u} f_{P_1}, {}^{a_n}_{a_m} f_{P_2})}{|Z_{P_1}^t(a_i)|} \quad (2)$$

where:

$$\mathsf{MF}^t({}^{a_v}_{a_u} f_{P_1}, {}^{a_n}_{a_m} f_{P_2}) = \begin{cases} M({}^{a_v}_{a_u} f_{P_1}, {}^{a_n}_{a_m} f_{P_2}) & \text{if} \begin{cases} t = 1, & (a_u = a_m) \vee (a_v = a_n) \\ & \vee (a_u = a_v \wedge a_m = a_n) \\ t \neq 1, & (a_u = a_m \wedge a_v = a_n) \end{cases} \\ 0 & \text{other cases} \end{cases}$$

For example, the service context matching between a_x and a_6 (Figure. 4) computed by Equation 2 is:

$$\begin{aligned} &\mathbb{MW}^3(G_{P_1}^{a_x}, G_{P_2}^{a_6}) = \\ &\frac{2}{3+1} \times (\frac{3}{3} \times \frac{M({}^{a_x}_{a_2} f_{P_1}, {}^{a_6}_{a_2} f_{P_2}) + M({}^{a_3}_{a_x} f_{P_1}, {}^{a_3}_{a_6} f_{P_2})}{2} \\ &+ \frac{2}{3} \times \frac{M({}^{e_1}_{a_2} f_{P_1}, {}^{e_3}_{a_2} f_{P_2}) + M({}^{a_4}_{a_3} f_{P_1}, {}^{a_4}_{a_3} f_{P_2})}{3} \\ &+ \frac{1}{3} \times \frac{M_p({}^{e_2}_{a_4} f_{P_1}, {}^{e_4}_{a_4} f_{P_2})}{2}) \\ &= \frac{2}{4} \times (\frac{0.66+0}{2} + \frac{2}{3} \times \frac{1+1}{3} + \frac{1}{3} \times \frac{1}{2}) = 0.47 \end{aligned}$$

Similarity between services is inferred by their composition context matching. Based on this similarity, we build a query which assist process designer to search for similar services. This query is presented in the next section.

III. Querying Services in Service-based Processes

In this section, we present a query language that is used to find similar services for a selected position in a service-based process based on the composition context matching. We present the query language in section III-A and its execution in section III-B.

A. Query's grammar

The query language in our approach consists of three parameters, which are: *an associated service*, *connection constraints*, and *a radius*. The associated service is the service whose composition context is taken into account to match

with other contexts. Connection constrains are services or connection flows to be included/excluded to filter the query's results. The radius is the number of connection layers taken into account for the composition context matching. It specifies the largeness of the considered composition contexts.

We present our proposed query's grammar using the Extended Backus-Naur Form[1]. We use ';' to separate the input parameters; '(' and ')' to separate query's constrains; '<' and '>' to group services or connection flows; '[' and ']' to separate a connection flow and its ending services. We use '+' and '-' signs to include/exclude constraints; and '|' sign for multiple choice operator. Details of the grammar are presented in Table I.

1.	Query	::=	ServiceID,':',[Constraint],':',Radius;
2.	ServiceID	::=	Character,{Character|Digit};
3.	Constraint	::=	('+'|'-')Term | Constraint,'|',Term;
4.	Term	::=	Item | Term,'+',Item | Term,'-',Item;
5.	Item	::=	ServiceID| ConFlow | '(',Constraint,')';
6.	ConFlow	::=	'<',[ServiceID],'[',FlowString,']',[ServiceID]'>';
7.	FlowString	::=	ConElement,{ConElement};
8.	ConElement	::=	'sequence' | 'AND-split' | 'AND-join' | 'OR-split' | 'OR-join' | 'XOR-split' | 'XOR-join' | 'event-based-gateway'| 'message-catching';
9.	Radius	::=	DigitNotZero,{Digit};
10.	Character	::=	'a' | 'b' | 'c' | 'd' | 'e' | 'f' | 'g' | 'h' | 'i' | 'j' | 'k' | 'l' | 'm' | 'n' | 'o' | 'p' | 'q' | 'r' | 's' | 't' | 'u' | 'v' | 'w' | 'x' | 'y' | 'z';
11.	DigitNotZero	::=	'1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9';
12.	Digit	::=	'0' | DigitNotZero;

TABLE I: Query grammar

The query grammar is explained as follows:

- The query is defined in line 1 with three parameters separated by ':'. The constraint is optional. It can be defined to filter the query result. It can also be eliminated if we want to execute only the composition context matching without filtering.
- In line 2, the service identifier is defined as a string of characters or digits. It has to start by a character.
- In line 3, we define constraints. A constraint can be an included/excluded service or connection flow. It can also include different items and operators. Operators in a constraint can be OR, INCLUDE, EXCLUDE. We use '|', '+' and '-' to specify these operators. Consequently, we define a constraint as a 'Term' or another constraint with '|' operator.
- In line 4 we define a 'Term' as an 'Item' or another constraint with the '+' and '-' operators.
- The 'Item' is defined in line 5. It can be a service or a connection flow that is included/excluded in the query. It can also be another constraint which is grouped by '(' and ')'. Definitions in line 3, 4 and 5 allow specifying a constraint with multiple services, connection flows, operations and grouped conditions.
- In line 6, we define a connection flow which is presented within '<' and '>' signs. It includes a string of connection elements that connects two services.
- The string of connection elements is defined in line 7. It includes at least a connection element which is defined in line 8.
- In line 9, we define the radius as a natural number greater than 0.
- Finally, lines 10, 11, 12 define literal characters and digits.

For example, during the design of train reservation process (Fig. 2), if the process designer wants to find services that have similar context to 'Search trains' (a_1) within the first zone, he can make a query as: a_1::1. If he wants to find services that are similar to 'Search trains' but not followed by 'Present alternatives', he will exclude the connection flow connecting 'Search trains' to 'Present alternatives' in its first layer. The query will be: a_1:-(<a_1[sequence]a_2>):1. In the case that he wants to know possible payment methods, he may select 'Process payment' service (a_3) and add a constraint which does not include a 'sequence' that connects to 'Send confirmation' service (a_4). He may also widen the considered composition contexts in two layers to get more results. Consequently, his query is as follows: a_3:-(<['sequence']a_4>):2. In another context, if he wants to know existing services that are similar to 'Search trains' (a_1), and include the services 'Request payment Info.' (a_8) and 'Process payment' (a_3) but exclude the AND-join connection between them, his query is as follows: a_1:+a_8+a_3-(<a_8['AND-join']a_3>):5.

B. Query's execution

In general, the procedure of getting results by using our query is as follows:

1) We capture the composition context of the associated service. The largeness of the composition context is specified by the radius parameter.
2) We match the composition context of the associated service to others in other service-based processes.
3) We refine the matching result by selecting only services whose composition contexts satisfy the query's constraints.
4) We sort the selected services based on the matching values and pick up top-N services[2] for the response.

IV. Implementation & Experiments

A. Implementation

To validate our approach as a proof of concept, we develop an application that can assist the process designer to find services using our query language. The application is developed and integrated into the Signavio[3], which is a business process modeling platform. Users are invited to use our application at http://www-inf.it-sudparis.eu/SIMBAD/tools/WebRec/[4].

The screen shot of the application is shown in Fig. 5. The areas 1, 2 and 3 show the BPMN elements for design, canvas and property of the selected element. They are provided by

[1]the EBNF is originally developed by Niklaus Wirth [23] and the EBNF standard is adopted by ISO, no. ISO/IEC 14977

[2]N can be flexibly tuned by the process designer

[3]Academic version at: http://www.signavio.com/en/academic.html and the open source at: http://code.google.com/p/signavio-core-components/

[4]Tutorial at: http://www-inf.it-sudparis.eu/SIMBAD/tools/WebRec/tutorial.html and video demo at: http://www-inf.it-sudparis.eu/SIMBAD/tools/WebRec/demo.html

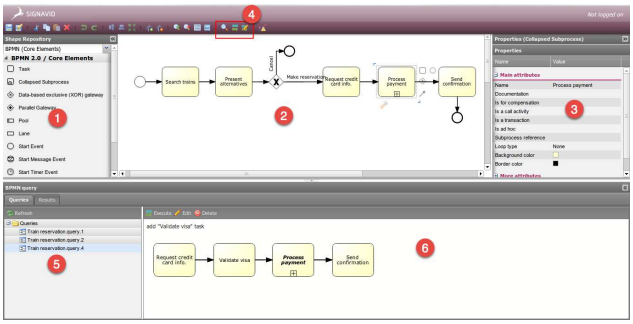

Fig. 5: Application's screen-shot

the Signavio platform. We added the areas 4, 5, and 6 into Signavio. Area 4 contains the buttons for activating the query design and execution, area 5 shows the list of historical queries for faster querying and area 6 shows the query design and results.

The composition context matching is executed beforehand. Then, query constraints are applied on the matching result to filter unrelated services.

B. Experiments

We performed experiments on a large public collection of business processes. Our goal is two fold: (i) to evaluate the feasibility of our proposed query and (ii) to measure its efficiency of our algorithm. Details of the dataset and experiments are given as following.

1) Dataset: The dataset used in our approach is shared by the Business Integration Technologies (BIT) research group[5] at the IBM Zurich Research Laboratory. It was presented in [14]. It contains business process models designed for financial services, telecommunications, and other domains. It is presented in XML format following BPMN 2.0 standard. Each XML file stores the data of a business process, including elements' IDs, service names, and the sequence flows between elements. The dataset consists of 560 business processes with 6363 services. There are 3781 different services in which 1196 services appear in more than one process. In average, there are 11.36 services, 2.59 'start' elements, 3.42 'end' elements, 18.96 gateways (including parallel, exclusive and inclusive gateways) and 46.81 sequence flows in a process .

2) Results: In the first experiment, we target to evaluate the *feasibility* of our service querying approach. To do so, we evaluate the number of services in the dataset that can be retrieved by querying the composition contexts within a given radius. We set $radius = 1$ and match the composition context graphs of all services in the repository using the proposed computation. We obtain 2938 (77.7%) services that match at least one service with a matching value greater than 0. It means that our approach can provide queried results for a majority services as we can retrieve similar composition contexts for more than 3/4 number of services.

In the second experiment, we evaluate the efficiency of our approach based on Precision values. We also compare our approach with an approach that query services randomly. As our approach takes into account composition contexts

[5]http://www.zurich.ibm.com/csc/bit/

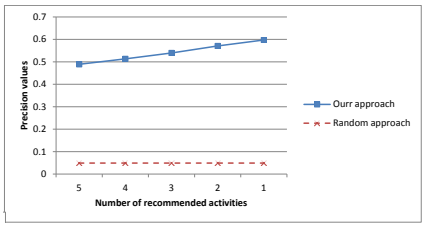

Fig. 6: Precision values with radius = 1

instead of service identifiers, we consider service identifiers as ground-truth data in computing Precision. Concretely, consider a selected service a in a process P. Assume that a appears in n processes. Also assume that the query of services that are relevant to this selected position returns a list of l services, in which $t(t \leq l)$ services are a. Precision of this query is given by Equation 3.

$$Precision = \frac{t}{l}; \tag{3}$$

In our experiment, we tune the number of queried services for each position from 5 to 1. We take into account only *the first zone* in our queries. To ignore the noise of the irrelevant processes, we compute Precision for only the services that appear in at least 10 processes. Consequently, 29 services and 267 processes are considered for our experiment.

Figure 6 shows the average Precision values obtained in our experiment. It shows that our queries achieved good Precision values. The Precision values increase when the number of queried services decreases. This means that the relevant services mostly appear at the top of the returned list. In other words, when we reduce the number of services in the returned list, the result shows that our approach are more focused and precise. It also shows that our approach is much more efficient than a random approach.

V. Related Work

Recently, there are many efforts on finding advanced solutions for process querying. Authors in [1], [19] proposed and developed a business process querying language, named BPMN-Q. The BPMN-Q helps to retrieve partial process fragments that start from a given activity (corresponding to service in our approach) and end by another given activity. Their query, however, always requires two activities as input and they have not yet dealt with the structural constrains, i.e. included/excluded activities and connection flows.

Momotko et. al. [18] proposed a query language to retrieve the status of service instance during the process execution time. Meanwhile, Beeri et. al. [2], [3], [4] proposed a query language for business process monitoring. Different from them, our approach can be applied in both design and execution phases. In addition, we exploit the relations between services in the design instead of the attributes of a service instance.

Deutch et. al. [8] were inspired by [2] and proposed a model for querying the structural and behavioral properties of business processes. While [9] proposed a query language for process specifications and Markovic et. al [17] proposed an ontoloty-based framework for process querying. In our approach, we focus on the service's 'behavior', i.e. relations

between services, instead of the process 'behavior', and we match the service (composition context) graphs instead of the process graphs.

Other approaches that aim at facilitating the process design without building a query language includes business process searching [24], [10] and process similarity measuring [21], [11], [16], [12], [13]. Their target is to help process designers find similar processes to the one they are designing. Related to them, our work also helps to facilitate process design. However, instead of comparing process models, we focus on comparing services in process models.

In a recent work [7], [6], [5], we recommended a list of services having a similar context graph in a process/service composition. Whereas in this work, we provide a graphical user interface and a context free grammar language to the process designer to specify his queries. Compared to our previous work, the process designer is able now to specify new constraints over the composition context using a defined query, and to view the corresponding results. This querying mechanism is richer and more useful than recommending simple services.

VI. Conclusion

In this paper, we present an original approach that can assist process designers during the design time and execution time. We introduce the service composition context and propose an algorithm to compute the similarity between services based on the composition context matching. We also propose a query language that can help to find services that are similar to selected services in a process. Experimental results show that our approach is feasible and efficient in querying services.

Due to the general limitation of public business process datasets, which provide only elements' identifier and services' names without any further information such as the one we used in our experiment, the validation of our approach so far is done with only the perfect match of services' names. However, our approach can be easily improved to deal with other comparison metrics and the validation can be extended for the imperfect matching. For example, we can take into account neighbors if their similarities on another comparison metric are greater than a certain threshold.

In our future work, we intend take into account the other service properties such as service description and pre-/post-conditions to refine the matching algorithm. We also aim at extending our approach to extract the hidden knowledge existing in business processes logs for another quality input.

References

[1] A. Awad, A. Polyvyanyy, and M. Weske. Semantic querying of business process models. In *Proceedings of the 2008 12th International IEEE Enterprise Distributed Object Computing Conference*, pages 85–94, Washington, DC, USA, 2008. IEEE Computer Society.

[2] C. Beeri, A. Eyal, S. Kamenkovich, and T. Milo. Querying business processes. In *Proceedings of the 32nd international conference on Very large data bases*, VLDB '06, pages 343–354. VLDB Endowment, 2006.

[3] C. Beeri, A. Eyal, T. Milo, and A. Pilberg. Monitoring business processes with queries. In *Proceedings of the 33rd international conference on Very large data bases*, VLDB '07, pages 603–614. VLDB Endowment, 2007.

[4] S.-M.-R. Beheshti, B. Benatallah, H. Motahari-Nezhad, and S. Sakr. A query language for analyzing business processes execution. In S. Rinderle-Ma, F. Toumani, and K. Wolf, editors, *Business Process Management*, volume 6896 of *Lecture Notes in Computer Science*, pages 281–297. Springer Berlin / Heidelberg, 2011.

[5] N. N. Chan, W. Gaaloul, and S. Tata. Composition context matching for web service recommendation. In H.-A. Jacobsen, Y. Wang, and P. Hung, editors, *IEEE SCC*, pages 624–631. IEEE, 2011.

[6] N. N. Chan, W. Gaaloul, and S. Tata. Context-based service recommendation for assisting business process design. In C. Huemer and T. Setzer, editors, *EC-Web*, volume 85 of *Lecture Notes in Business Information Processing*, pages 39–51. Springer, 2011.

[7] N. N. Chan, W. Gaaloul, and S. Tata. Assisting business process design by activity neighborhood context matching. In C. Liu, H. Ludwig, F. Toumani, and Q. Yu, editors, *ICSOC*, volume 7636 of *Lecture Notes in Computer Science*, pages 541–549. Springer, 2012.

[8] D. Deutch and T. Milo. Querying structural and behavioral properties of business processes. In *Proceedings of the 11th international conference on Database programming languages*, DBPL'07, pages 169–185, Berlin, Heidelberg, 2007. Springer-Verlag.

[9] D. Deutch and T. Milo. A structural/temporal query language for business processes. *J. Comput. Syst. Sci.*, 78(2):583–609, 2012.

[10] R. Dijkman, M. Dumas, and L. Garcia-Banuelos. Graph matching algorithms for business process model similarity search. In *BPM*, pages 48–63, 2009.

[11] R. M. Dijkman. A classification of differences between similar business processes. In *11th IEEE International Enterprise Distributed Object Computing Conference, 15-19 October 2007, Annapolis, Maryland, USA*, pages 37–50, 2007.

[12] B. Dongen, R. Dijkman, and J. Mendling. Measuring similarity between business process models. In *Proceedings of the 20th international conference on Advanced Information Systems Engineering*, CAiSE '08, pages 450–464, Berlin, Heidelberg, 2008. Springer-Verlag.

[13] M. Ehrig, A. Koschmider, and A. Oberweis. Measuring similarity between semantic business process models. In *APCCM '07*, pages 71–80, 2007.

[14] D. Fahland, C. Favre, B. Jobstmann, J. Koehler, N. Lohmann, H. Völzer, and K. Wolf. Instantaneous soundness checking of industrial business process models. In *7th BPM*, pages 278–293, 2009.

[15] V. Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*, 10:707, 1966.

[16] C. Li, M. Reichert, and A. Wombacher. On measuring process model similarity based on high-level change operations. In *ER '08*, pages 248–264, 2008.

[17] I. Markovic, A. Costa Pereira, D. Francisco, and H. Muñoz. Service-oriented computing - icsoc 2007 workshops. chapter Querying in Business Process Modeling, pages 234–245. Springer-Verlag, Berlin, Heidelberg, 2009.

[18] M. Momotko and K. Subieta. Process query language: A way to make workflow processes more flexible. In *Advances in Databases and Information Systems*, volume 3255 of *Lecture Notes in Computer Science*, pages 306–321. Springer Berlin / Heidelberg, 2004.

[19] S. Sakr and A. Awad. A framework for querying graph-based business process models. In *Proceedings of the 19th international conference on World wide web*, WWW '10, pages 1297–1300, New York, NY, USA, 2010. ACM.

[20] W. Van Der Aalst. Business process management: A personal view. *Business Process Management Journal*, 10(2):248–253, 2004.

[21] W. van der Aalst, A. de Medeiros, and A. Weijters. Process equivalence: Comparing two process models based on observed behavior. In *BPM*, volume 4102, pages 129–144, 2006.

[22] W. van der Aalst, A. H. M. T. Hofstede, and M. Weske. Business process management: A survey. In *Proceedings of the 1st International Conference on Business Process Management*, pages 1–12. Springer-Verlag, 2003.

[23] N. Wirth. What can we do about the unnecessary diversity of notation for syntactic definitions? *Commun. ACM*, 20(11):822–823, Nov. 1977.

[24] Z. Yan, R. Dijkman, and P. Grefen. Fast business process similarity search with feature-based similarity estimation. In *OTM (1)*, pages 60–77, 2010.